Programming Assignment № 2

Mustafa Sercan Amac , Hacettepe University 01/01/2014

Feature Extraction

I used SIFT to extract features from images using Opencv's SIFT module. I extracted keypoints and descriptions using detect and compute function from this module. This function takes an image as an input and outputs keypoints and descriptions for that image. None corresponds to outimage parameter which is not needed in python.

Listing 1: Feature extraction code .

```
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
```

Feature Matching

For feature matching i used BfMatch module and i used knnMatch function with k = 2. And then i filtered out good matches to obtain better results. I also plotted keypoints for pair of images and match points.

Listing 2: Feature Matching Code

```
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
good = []
for m, n in matches:

    if m.distance < 0.65 * n.distance:
        good.append(m)
#good = bf.match(des1, des2)

keypoints = cv2.drawKeypoints(img1,kp1,None,flags=2)
keypoints2 = cv2.drawKeypoints(img2,kp2,None,flags=2)

match_points = cv2.drawMatches(img1, kp1, img2, kp2, good, None, flags=2)
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
ax1.imshow(keypoints)
```

```
ax2 = fig.add_subplot(1,2,2)
ax2.imshow(keypoints2)
ax1.title.set_text("Left  image with keypoints")
ax2.title.set_text("Right image with keypoints")
plt.pause(0.01)
plt.imshow(match_points)
plt.pause(0.01)
```

Finding Homography

I used RANSAC technique to find the homography matrix. The function takes keypoints,matches, number of points to choose, number of iterations and the threshold value. The function first chooses n random numbers and using them i created the matrix to find homography matrix. I used SVD method from numpy to estimate the homography. The error metric i used is euclidian distance.

Listing 3: Find Homography matrix

```
def findHomography(matches, kp1, kp2, num=8, num_iters=10000, threshold=1):
    best = 0
    best_H = []
    for i in range(num_iters):
        indices = np.random.choice(len(matches), num, replace=True)
        temp_matches = []
        for ind in indices:
            temp_matches.append([kp1[ind], kp2[ind]])
        P = getSVDmatrix(temp_matches)
        A, B, C = np.linalg.svd(P)
        H = C[-1, :].reshape((3, 3))

        H = (1 / H.item(8)) * H

        H_query = np.dot(np.hstack((kp1, np.ones((kp1.shape[0], 1)))), H)
        H_query = H_query[:, :-1]
        p1 = np.hstack((kp2, np.ones((kp2.shape[0], 1))))
        p2 = np.hstack((kp1, np.ones((kp1.shape[0], 1))))
        p1hat = np.dot(H, p2.T).T
        W = p1hat[:, -1].reshape(p1hat.shape[0], 1).repeat(3, axis=-1)
        p1hat = p1hat / W
        diffs = np.linalg.norm(p1hat - p1, axis=1)
        count = len(np.where(diffs < threshold)[0])
        if count > best:
            best = count
            best_H = H
```

```python
            #print(best)

    return best_H
```

Correcting Homography

After obtaining the homography, the problem is lots of information is lost when i warped the perspective. The solution is to determine where the corners point will lie in the transformed image and translating the image by changing the offset indices in Homography matrix by the minimum value of the obtained minimum x and minimum why if they are negative.

Listing 4: CorrectHomography

```python
def getHomCorrection(width, height, H):
    P = [[0, width, width, 0],
         [0, 0, height, height],
         [1, 1, 1, 1]]
    A = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
    pus = np.dot(H, P)
    pus[0] = pus[0] / pus[2]
    pus[1] = pus[1] / pus[2]
    minx, miny = min(pus[0]), min(pus[1])
    if minx < 0:
        A[0,2] = -minx
    if miny < 0:
        A[1,2] = -miny
    best_A = np.dot(A, H)
    return best_A
```

Warp Perspective

To apply the homography matrix on a image, i first created an array of all possible indices and transformed these indices to homogenous coordinates then i divided them and made them cartesion coordinates. And then i rounded them because they correspond to subpixel coordinates. Then i created an empty array to create the translated image. And then i added a black region to the beginning of the image, by size of the other image's second dimension , so i can do direct pixel wise alignment to merge the images.

Listing 5: Warp Perspective

```python
def warpImage(H,img1,img2):
    width, height = img2.shape
    result = np.zeros((width, height))
    indices = np.indices((width, height))
    rows = indices[0].reshape(indices[0].shape[0] * indices[0].shape[1])
    cols = indices[1].reshape(indices[1].shape[0] * indices[1].shape[1])
    ones = np.ones(rows.shape[0])
    inds = np.vstack((rows, cols, ones))
    warped_inds = np.dot(np.linalg.inv(H), inds)
    warped_inds[0], warped_inds[1] = warped_inds[0] / warped_inds[2], warped_inds[1] / warped_inds[2]
    warped_inds = np.int32(np.rint(warped_inds[:2]))
    warped_inds = (warped_inds[0], warped_inds[1])
    nonzero_inds1 = np.where(np.logical_and(np.logical_and(np.logical_and((warped_inds[0] >= 0) ,(warped_inds[1] >= 0)) ,(warped_inds[0] < result.shape[0])), (warped_inds[1] < result.shape[1])))
    non_zero_rows = warped_inds[0][nonzero_inds1]
    non_zero_cols = warped_inds[1][nonzero_inds1]
    warped_inds = (non_zero_rows,non_zero_cols)

    img_inds = (np.int32(inds[0][nonzero_inds1]), np.int32(inds[1][nonzero_inds1]))
    result[img_inds] = img2[warped_inds]
    zeros = np.zeros((img1.shape[0],img1.shape[1]+20))
    result = np.hstack((zeros,result))
    return result.astype("uint8")
```

Finding an alignment between 2 image

To stitch 2 image we need to decide where to align these images. I used squared distance as metric and shifted the first image over the translated image and found the best alignment position.

Listing 6: Alignment

```python
def findAlignment(img1, img2):
    height, width = img2.shape
    best_width = 0
    best_sqd = 9999999
    widths = []
    sqds = []
    regions = []
    while width >= img1.shape[1]:
        test_region = img2[:, width - img1.shape[1]:width]

        sqd = np.sqrt(np.sum(np.square(test_region - img1)))
        sqds.append(sqd)
        widths.append(width)
        regions.append(test_region)
        if sqd < best_sqd:
            best_sqd = sqd
            best_width = width

        width = width - 1

    return best_width
```

Merging The Images

After finding the best alignment position we can merge 2 images in to one and form a panaroma.

Listing 7: Merge

```
def mergeImages(img1,img2,best_width):
    w = best_width
    x = img2.copy()

    p = x[:, w - img1.shape[1]:w].copy()

    p[:img1.shape[0], :img1.shape[1]] = img1

    res = np.hstack((p,x[:, w::]))
    return res
```

Sample Results

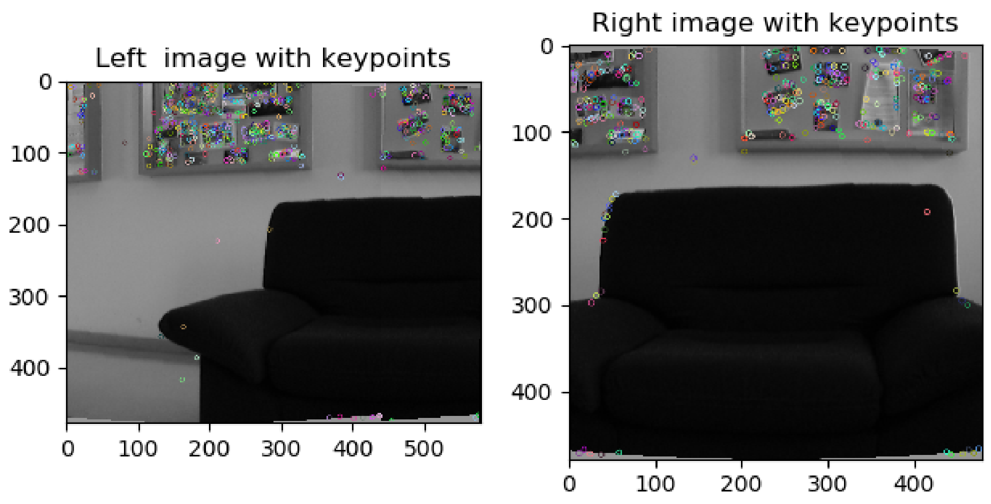

Figure 1: Sample feature points

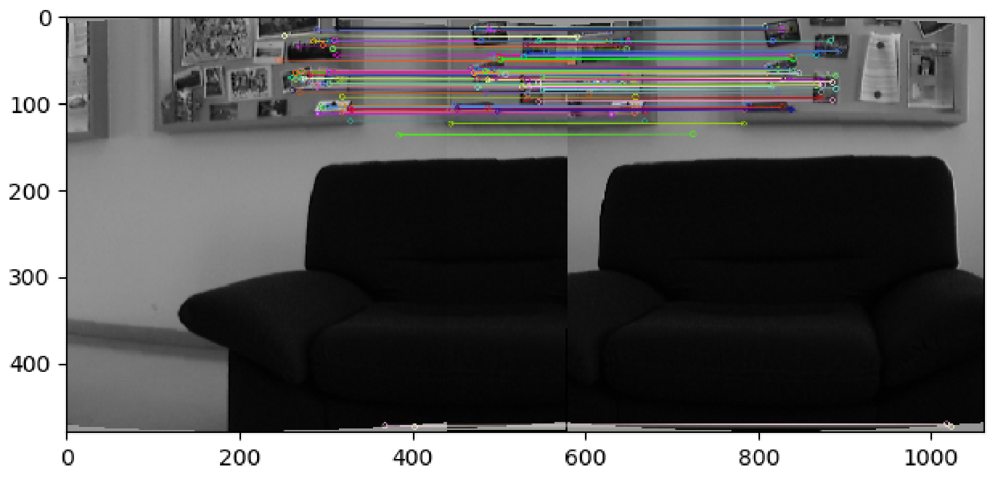

Figure 2: Sample feature match

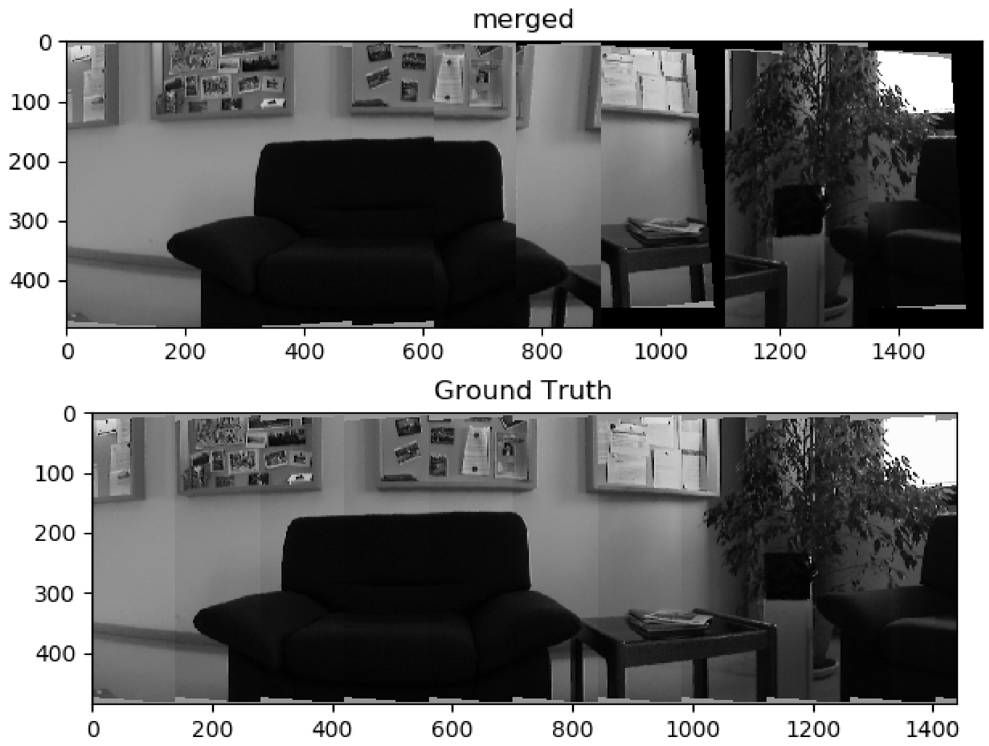

Figure 3: Sample panaroma

Comments About Results

The results i got was almost same with the ground truth panaroma in some images as can be seen on the samples. Since finding the homography is a s thocastic process with RANSAC and not every sub image has good matches i didn't expect to get good results but it was impressive. The sample I've shown is formed from 8 images. We can obtain better results with better alignment and feature matching techniques. A deep note I've figured out that instead of matching the new image with current panaroma it is better to match with the last image sized region.